Grouping Semantic

SoCIM, 10th of May 2017, London
Paweł Susicki, Jan Posiadała
{trzeci,janek}@tiger.com.pl
Implied group by

Implied group by is neat but, two following queries give (in neo4j) two different results:

```
create (:L {a:1,b:2,c:3})
create (:L {a:2,b:3,c:1})
create (:L {a:3,b:1,c:2})
```

### Query 1
```
match (x:L)
return x.a + count(*) + x.b + count(*) + x.c;
```

Query Results
```
+----------------------------------------+
<table>
<thead>
<tr>
<th>x.a + count(<em>) + x.b + count(</em>) + x.c</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
</tr>
<tr>
<td>8</td>
</tr>
<tr>
<td>8</td>
</tr>
</tbody>
</table>
+----------------------------------------+
3 rows
96 ms
```

### Query 2
```
match (x:L)
return x.a + x.b + x.c + count(*) + count(*);
```

Query Results
```
+----------------------------------------+
<table>
<thead>
<tr>
<th>x.a + x.b + x.c + count(<em>) + count(</em>)</th>
</tr>
</thead>
<tbody>
<tr>
<td>12</td>
</tr>
</tbody>
</table>
+----------------------------------------+
1 row
77 ms
```

No adjustment in TCK

Grouping Semantic
SoCIM, 10th of May 2017, London
Problem: to designate aggregation key (and aggregated expressions).

```
match (x:L)
return x.a + count(*) + x.b + count(*) + x.c;
```

Query Results

```
+----------------------------------------+
| x.a + count(*)  + x.b + count(*) + x.c |
| 8                                      |
| 8                                      |
| 8                                      |
+----------------------------------------+
```

Aggregation key: \((x.a, x.b, x.c)\) gives result of size 3

```
match (x:L)
return x.a + x.b + x.c + count(*) + count(*);
```

Query Results

```
+----------------------------------------+
| x.a + x.b  + x.c + count(*) + count(*) |
| 12                                     |
+----------------------------------------+
```

1 row
77 ms

Aggregation key: \((x.a + x.b + x.c)\) gives result of size 1
A heuristic algorithm of extraction component expression of grouping key, which fails in the given example due to some optimization operations applied on the query which are supposed to be equivalent, but they are not.
Grouping key is the tuple built from all variables (node, relationship, their properties or variables chained from the previous subquery) that appear outside of aggregate functions of a particular WITH/RETURN clause.

Aggregation key: all atomic („leaves”) expressions

\[(x.a, x.b, x.c)\]

i.e. for both queries left result – 3 records.
Used in: ingraph, AgenGraph.
Cons of Option 1 is shown with this example:

```
CASE
    WHEN false THEN x.a   Grouping key: (x.a,x.b)
    WHEN true  THEN x.b
END
```
Each item of the expression list in `WITH/RETURN` are forced to contain either:

(i) no aggregate function or a
(ii) single aggregate function at the outermost level

The grouping key is the tuple built from items of type (i)
(this is the approach in #188, #218).

which is another name for: how to hide explicit `group by`
Proposed approach

\[ a + \text{count}(*) + b \times \text{count}(*) + c \]

\[ a + 0 + b \times 1 + c \]
Let's think about a little bit more complicated expression:

```
WITH list[a..min(b)]
```

there we can see, \( \text{min}(y) \) influence \( l[x..] \) which changes shape of groups, so we want to deny using such constructions with message:

„*Please contact your personal algebraist*”
QUESTION 1

How to define

semantics for key selection (in implicit grouping expressions)

such a way, that

selected key is independent of value of an aggregating function

(i.e. replacing aggregating functions with constant values does not change groups selected by the keys).

PROPOSITION

Replace aggregating function with expression above up to binary operator by neutral value of the operator.

QUESTION 2

How to enhance the idea to all operators, especially not having neutral value (e.g.: index on list).
A & Q